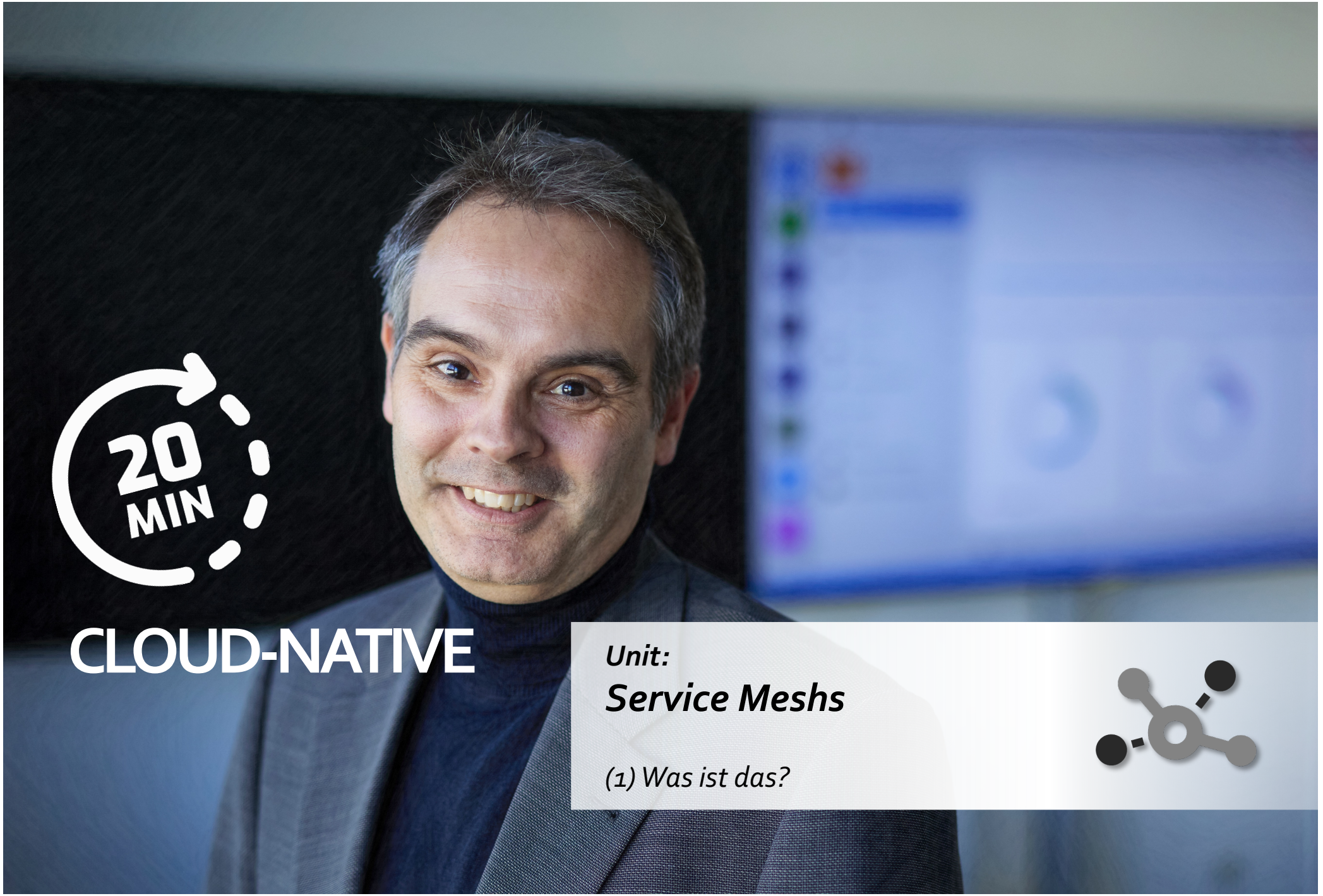


CLOUD NATIVE COMPUTING

CLOUD-NATIVE

Unit:

Service Meshs

(1) Was ist das?

creative commons

CLOUD NATIVE COMPUTING

Urheberrechtshinweise

Diese Folien werden zum Zwecke einer praktikablen und pragmatischen Nutzbarkeit im Rahmen der **CCo 1.0 Lizenz** bereitgestellt.

Sie dürfen die Inhalte also kopieren, verändern, verbreiten, mit eigenen Inhalten mixen, auch zu kommerziellen Zwecken, und ohne um weitere Erlaubnis bitten zu müssen.

Eine Nennung des Autors ist nicht erforderlich (aber natürlich gern gesehen, wenn problemlos möglich).

Diese Folien sind insb. für die Lehre an Hochschulen konzipiert und machen daher vom **§51 UrhG (Zitate)** Gebrauch.

Die CCo Lizenz überträgt sich nicht auf zitierte Quellen. Hier sind bei der Nutzung natürlich die Bedingungen der entsprechenden Quellen zu beachten.

Die Quellenangaben finden sich auf den entsprechenden Folien.

KAPITEL 13

Automatisierte Instrumentierung mittels Service Meshs

13.1 Konsolidierung von Telemetriedaten

13.2 Instrumentierung von Systemen

- Logging
- Monitoring
- Tracing

13.3 Automatisierte Instrumentierung

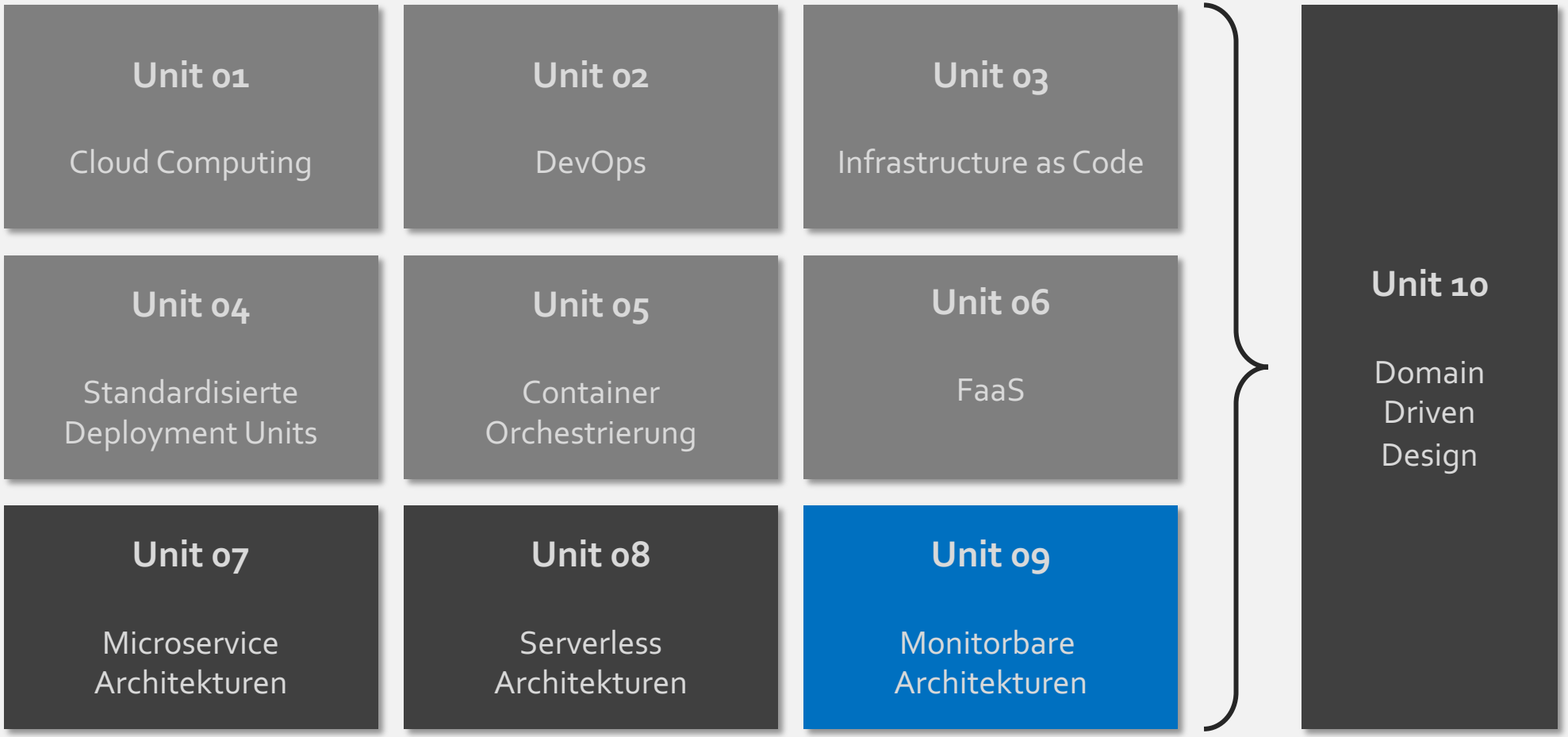
- Service Meshs
- Traffic-Management
- Resilienz
- Sicherheit
- Management und Analyse von Verkehrstopologien

13.4 Zusammenfassung

CLOUD NATIVE COMPUTING

INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls



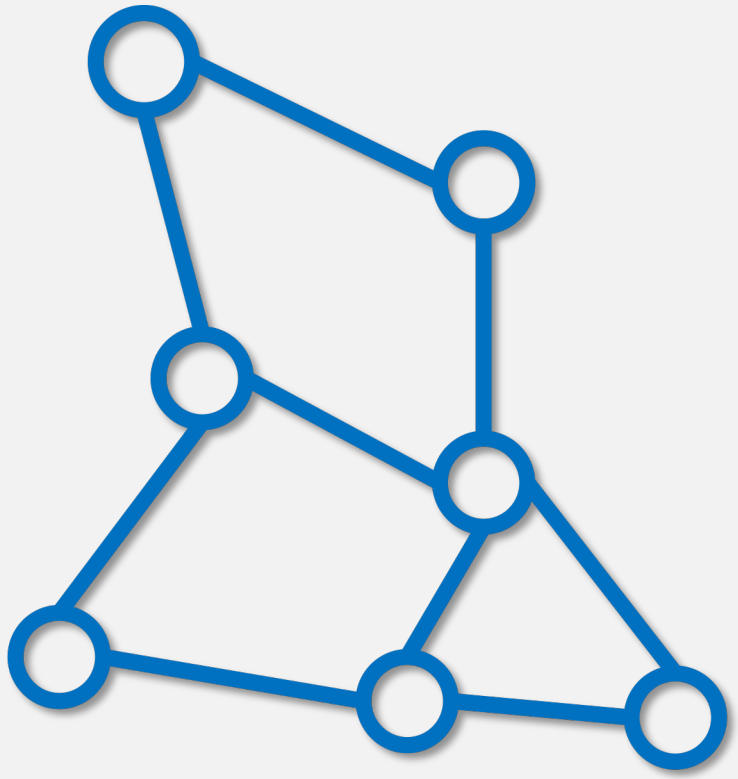
INHALTE

Was sind Service Meshs?

SMI-Standard

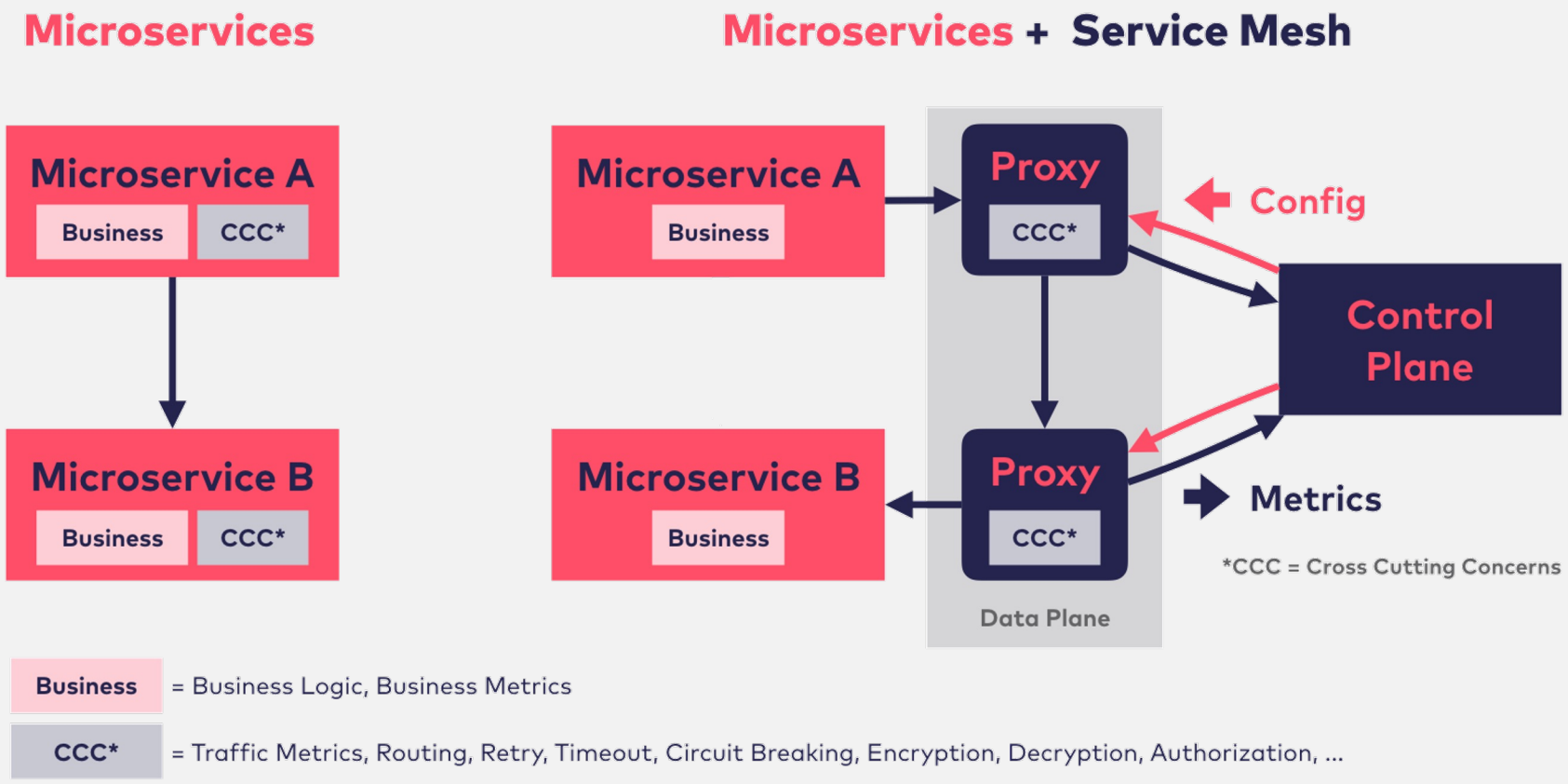
Funktionalitäten von Service Meshs am Beispiel von Istio

- Traffic Management
- Resilienz
- Sicherheit
- Beobachtbarkeit



SERVICE MESH

TLDR: Handhabung von Microservice Cross-Cutting-Concerns mittels des Sidecar-Patterns

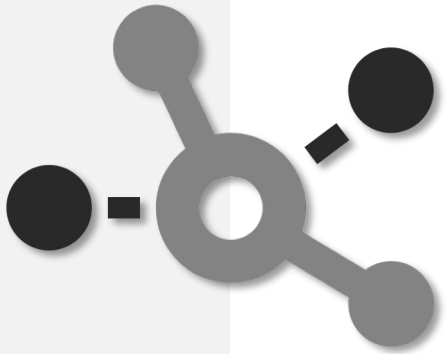


Bildquelle: servicemesh.es

SERVICE MESH

Was ist das?

- Ein Service Mesh ist eine **Netzwerkmanagement-Schicht** für verteilte Landschaften von unabhängigen Microservices
- Es handelt sich um eine Schicht von **Proxy-Servern**, die als Daten- und Steuerebene zwischen den Microservices dienen
- Ein Service Mesh kann verschiedene Funktionen für **Cross-Cutting-Concerns** von Microservice Architekturen bereitstellen, wie z.B.
  - Lastausgleich
  - Service-Discovery
  - Fehlerbehebung und Sicherheit
- Ein Service Mesh kann auch die **Service-to-Service-Kommunikation** verbessern, z.B. mittels:
  - Circuit Breaking
  - Retry-Logik
  - Timeout-Konfiguration
  - Fehlerbehebung



SERVICE MESH

Wofür braucht man das? Was kann das?

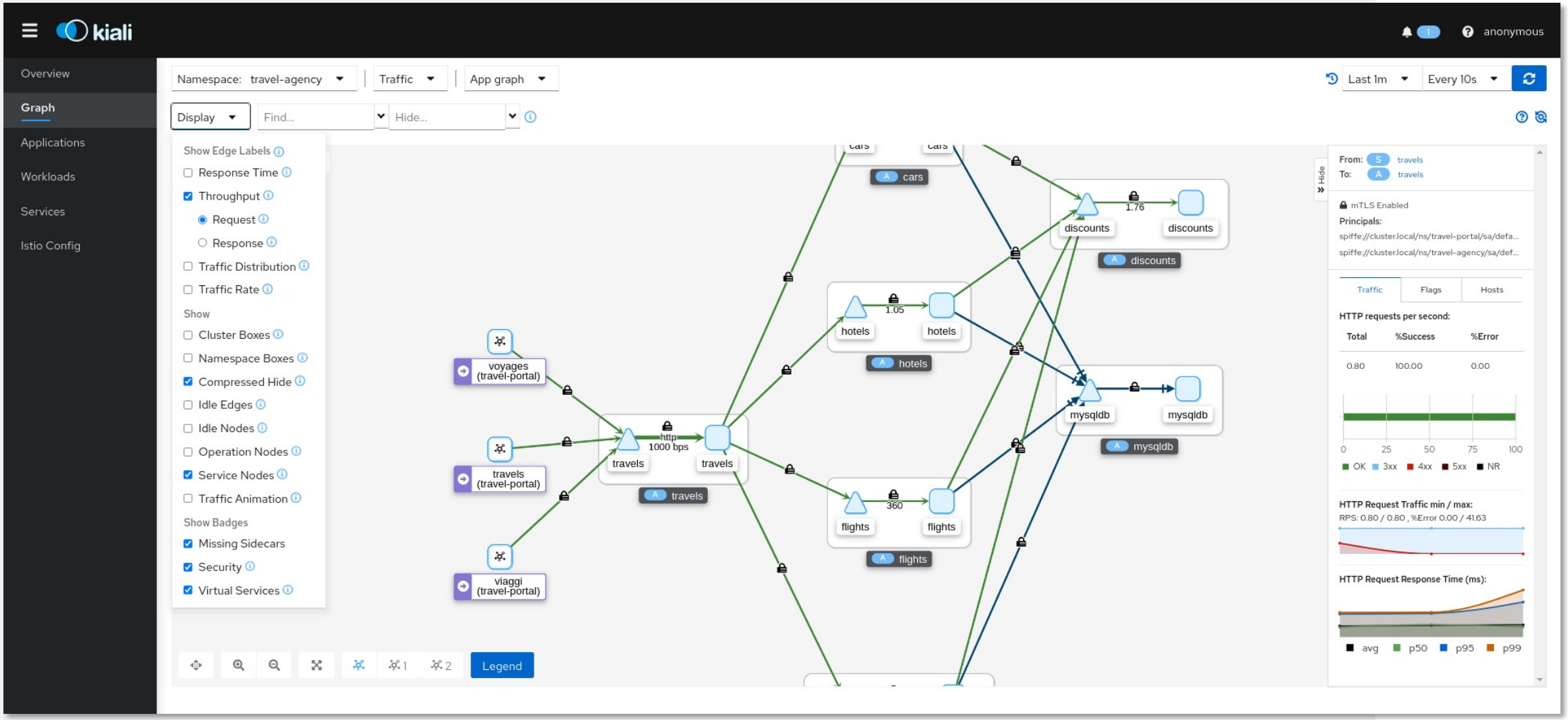
- Service Meshs sind entstanden, um die durch Microservice Architekturen entstehenden Herausforderungen besser zu managen zu können.
- In einer Microservices-Architektur kommunizieren einzelne Dienste untereinander, so dies sich diverse Herausforderungen hinsichtlich Zuverlässigkeit, Sicherheit und Skalierbarkeit ergeben

Ein Service Mesh bietet hierzu folgende Funktionalitäten

- **Service Discovery**
  Service Meshs bieten eine zentrale Service-Discovery-Funktionalität, die es ermöglicht, Dienste in einer dynamischen Umgebung zu finden.
- **Load Balancing**
  Last auf mehrere Instanzen eines Dienstes verteilen, um eine höhere Verfügbarkeit und Skalierbarkeit zu erreichen.
- **Sicherheit**
  Sicherheitsfunktionen wie Transport Layer Security (TLS), Authentifizierung, Autorisierung und Verschlüsselung
- **Observability**
  Detaillierte Einblicke in den Datenverkehr zwischen Diensten, um Probleme zu erkennen und Fehlerbehebung zu erleichtern
- **Service-to-Service-Kommunikation**
  Funktionen wie Service-Retry, Circuit-Breaking und Timeout-Konfigurationen die nicht in den Diensten selber implementiert werden muss

SERVICE MESH

Ermöglicht eine Form der Black-Box Observability



Bildquelle: kiali.io

SERVICE MESH

„Definition"

Grund für Service Meshs

- Die Entwicklung einzelner Microservices ist leicht
- Der Betrieb einer Microservice Architektur ist jedoch nicht trivial

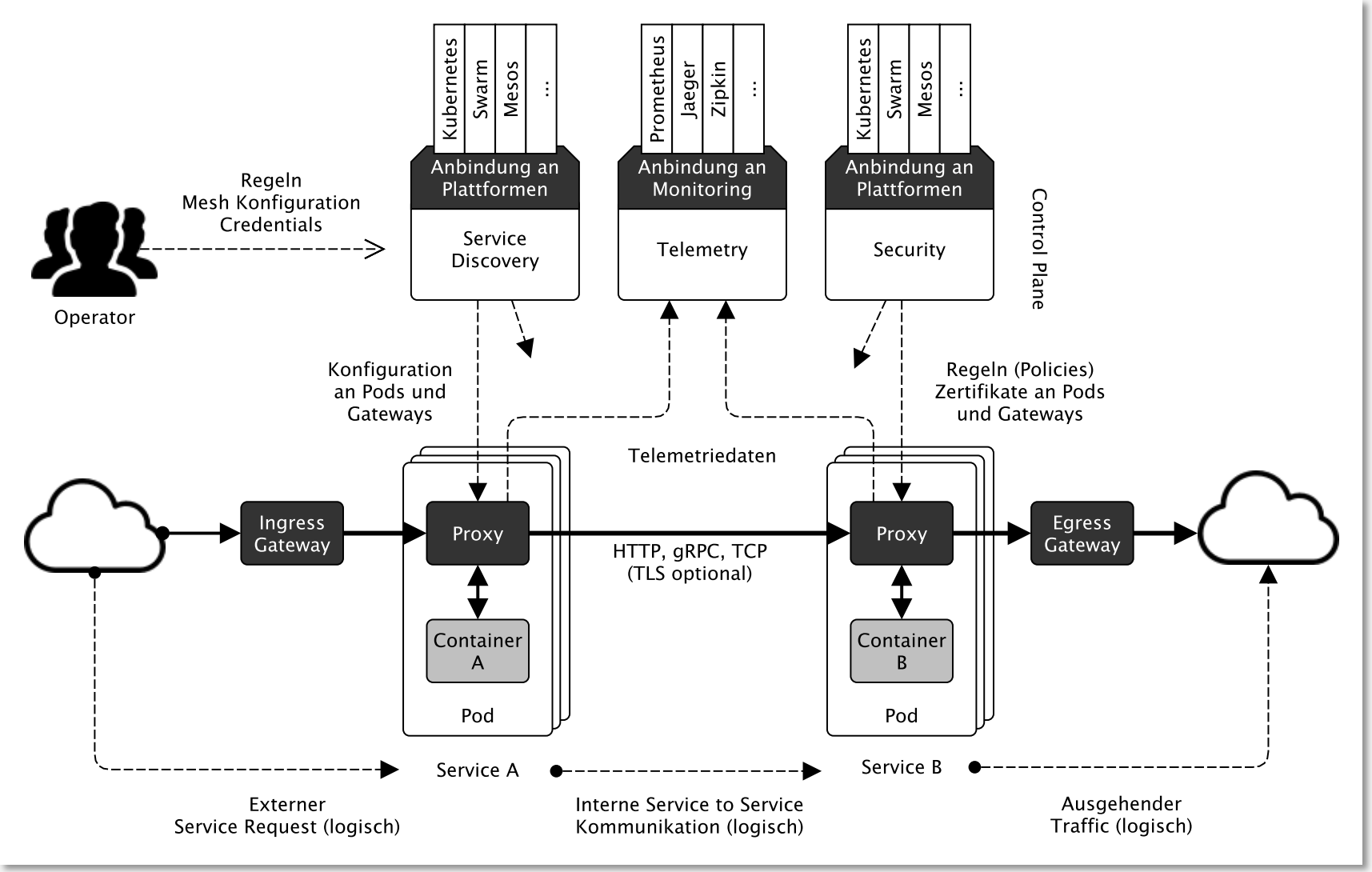
Technischer Ansatz:

- Kanalisierung von Netzwerkinteraktionen zwischen Microservices durch Proxies
- Proxies können dann Netzwerkverkehr verschlüsseln, analysieren, überwachen und routen

Ein Service Mesh erzeugt eine ***Netzwerkabstraktion*** für containerbasierte Applikationen und Dienste, um deren Management zu vereinfachen und Aspekte wie ***Traffic Management***, ***Zuverlässigkeit***, ***Beobachtbarkeit*** *und* ***Sicherheit*** innerhalb von Microservice Architekturen zu vereinfachen und ***querschnittlich*** zu isolieren.

SERVICE MESH

Architektur

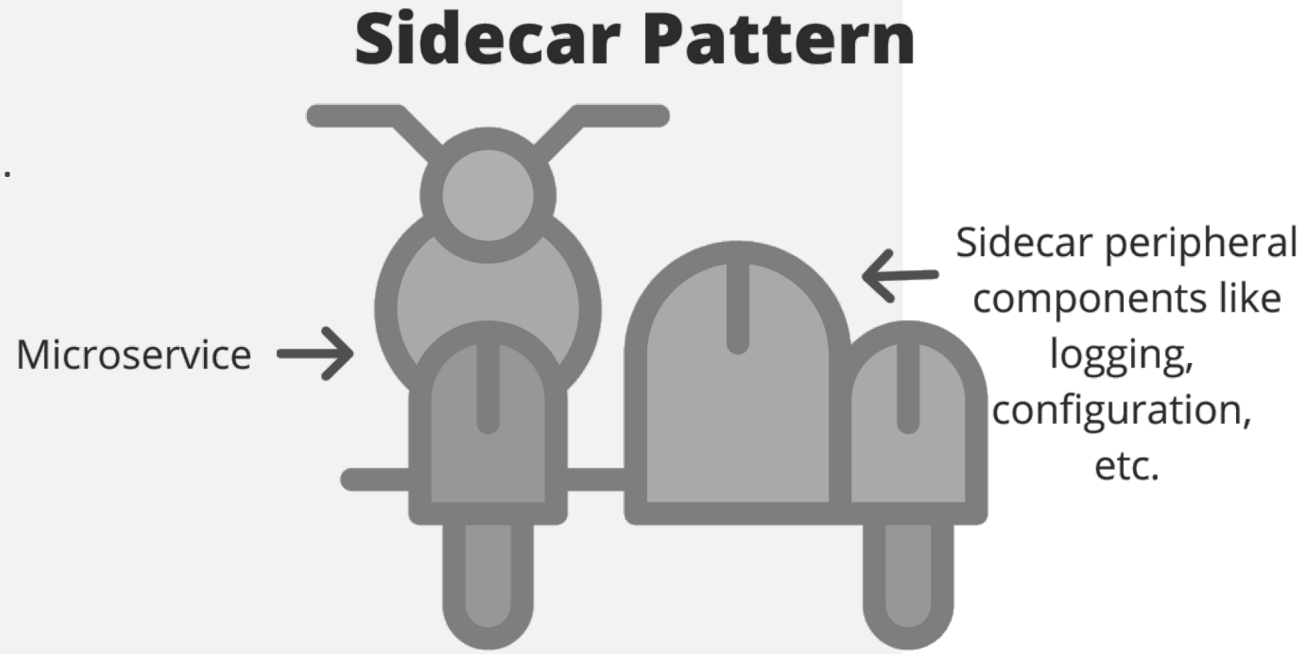


*Ein Service Mesh besteht aus einer Kontrollebene (**Control plane**), die **Proxies** (Sidecar Pattern) steuert. Die Proxies sind Serviceinstanzen zugeordnet (Sidecar Pattern) und spannen eine (ggf. verschlüsselte) Datenschicht (**Data plane**) für Services auf.*

SERVICE MESH

Sidecar-Pattern

- Das Sidecar Pattern ist ein **Entwurfsmuster** und besteht aus zwei Komponenten: einer Hauptanwendung und einem separaten, aber eng gekoppelten Prozess, der als Sidecar bezeichnet wird.
- Das Sidecar-Pattern funktioniert, indem es den **Netzwerkverkehr** zwischen der Hauptanwendung und anderen Anwendungen **abfängt**, die über das Netzwerk kommunizieren.
- Der Sidecar-Prozess ist ein **zusätzlicher Container**, der in demselben Pod wie die Hauptanwendung ausgeführt wird und den Datenverkehr mit anderen Anwendungen abfängt und verarbeitet.
- Der Sidecar kann so **zusätzliche Funktionalitäten** wie Lastausgleich, Service Discovery, Sicherheit und Überwachung außerhalb der eigentlichen Anwendung bereitstellen.



SERVICE MESH

Wesentliche Komponenten und deren Verantwortlichkeiten in einem Service Mesh

Proxy

- Jeder Dienst kommuniziert über einen **dedizierten Proxy**
- Die gesamte Kommunikation im Mesh erfolgt über diese Proxies
- Proxies können so das **Kommunikationsverhalten anreichern** und z.B. Tracing- und Monitoring-Daten sammeln
- Proxies lassen sich von Orchestrierungs-Frameworks verwalten, um so z.B. bei jedem Deployment **automatisch platziert** zu werden

Control plane

- **Verwaltung** von Routen und Regeln
- **Konfiguration** von
  - Timeouts, Retries, etc.
  - Service Discovery
  - Access Control
  - Policy Control
  - Telemetriedaten

Data plane

- **Ein-** (ingress) und **ausgehende** (egress) Kommunikation
- **Service-to-Service Kommunikation**
- Kommunikation über Proxies
- **Transparent** für verwaltete Dienste

SERVICE MESH

Vorteile und Nachteile des Sidecar-Pattern

Vorteile

- **Bessere Modularität**
  Es lassen sich zusätzliche Funktionen zu einer Anwendung hinzuzufügen, ohne die Anwendung selbst zu ändern.
- **Bessere Entkopplung**
  Der Sidecar-Prozess ist mit der Hauptanwendung gekoppelt, aber unabhängig davon. So kann der Sidecar-Prozess unabhängig aktualisiert werden, ohne die Hauptanwendung zu beeinträchtigen.
- **Höhere Flexibilität**
  Es lassen sich verschiedene Sidecar-Implementierungen zu verwenden, um unterschiedliche Funktionalitäten wie Sicherheit oder Überwachung bereitzustellen.

Nachteile

- **Erhöhte Komplexität**
  Führt zu höherer Komplexität der Anwendungsarchitektur, da es zusätzliche Prozesse oder Container erfordert
- **Erhöhter Ressourcenverbrauch**
  Erhöht den Ressourcenverbrauch (zusätzliche Prozesse oder Container)
- **Erhöhte Netzwerklatenz**
  Netzwerkverkehr wird über zusätzliche Schichten und Prozesse geleitet
- **Eingeschränkte Kompatibilität**
  Anwendungen müssen auf eine bestimmte Weise entwickelt und konfiguriert werden. Dies kann Einschränkungen bei Frameworks oder Bibliotheken bedeuten.

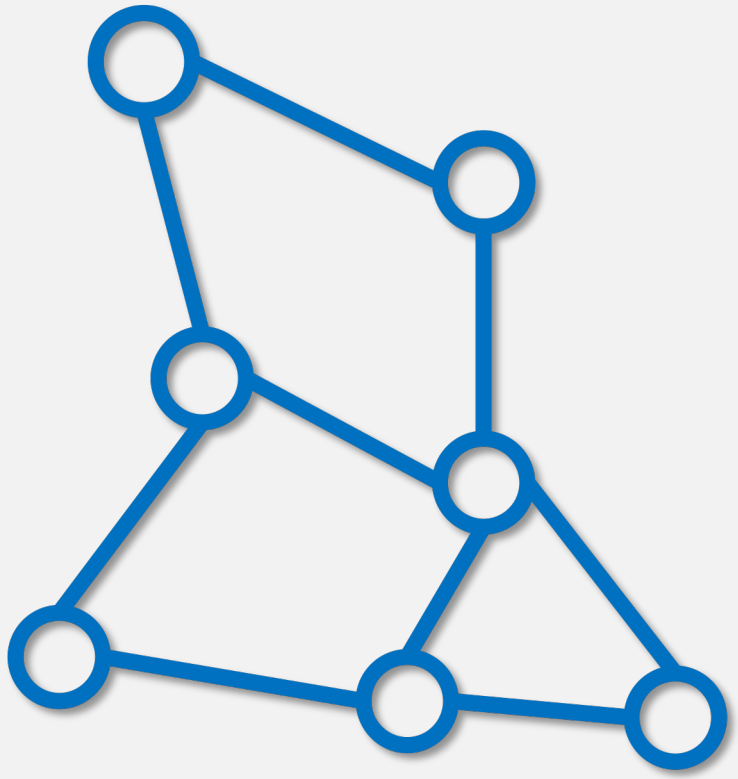
AUSBLICK

Was sind Service Meshs?

SMI-Standard

Funktionalitäten von Service Meshs am Beispiel von Istio

- Traffic Management
- Resilienz
- Sicherheit
- Beobachtbarkeit



KONTAKT

Disclaimer

Nane Kratzke

+49 451 300-5549

nane.kratzke@th-luebeck.de

kratzke.mylab.th-luebeck.de